INFERENCING DATA TYPES OF MESSAGE COMPONENTS

INVENTORS
ABHISHEK CHAUHAN
RAJIV MIRANI
PRINCE KOHLI
NAMIT SIKKA

BACKGROUND

Field of Invention

[0001] The present invention relates generally to data type inferencing of scalar objects, and more particularly, to one application of inferencing data types of message components to generate rules which block illegitimate messages from passing through application proxies and gateways.

Background of the Invention

[0002] Corporations are rapidly deploying web-based applications to automate business processes and to facilitate real-time interaction with customers, business partners and employees. Highly vulnerable to malicious hackers, web applications provide an entry point through which sensitive data can be accessed and stolen. Given the vulnerability of web applications, establishing a web application protection is critical for any enterprise that is exposing sensitive data or transaction systems over the Internet.

[0003] Firewalls are an essential component in a corporate entity's network security plan. They represent a security enforcement point that separates a trusted network from an untrusted network. Firewalls determine which traffic should be allowed and which traffic should be disallowed based on a predetermined security policy.

[0004] Firewall systems designed to protect web applications are known. They are commonly implemented as application proxies or application gateways. An application proxy is an application program that runs on a firewall system between two networks and acts as an intermediary between a web client and a web server. When client messages are received at the firewall, the final server destination address is determined by the application proxy software. The application proxy translates the address, performs additional access control checking, and connects to the server on behalf of the client. An application proxy authenticates users and determines whether user messages are legitimate.

[0005] Application proxies and gateways, however, are not designed to determine whether messages and their components are of a proper data type. This functionality is expected to be performed by web applications. For example, when an XML message is sent across a network, a web application extracts components of the message, typically in the form of field name-value pairs. The web application then determines whether the values are of an expected data type using type-checking rules as is well known in the art. If the web application determines that a value is not of a valid data type, as specified by the application, the web server does not process the message.

[0006] Existing methods that rely on verifying data types of message components, however, have a number of shortcomings. First, since application proxies and gateways are not configured to verify the data types of message components, messages containing components of an invalid data type will still be allowed to pass through application proxies and/or security gateways. Since some of these messages are more likely to represent malicious attacks rather than legitimate requests, failure to verify data types by application proxies and gateways results in allowing illegitimate traffic to enter web servers.

[0007] Secondly, the schemes used by a web server to verify data types of message components are manually provided by developers. Therefore, security policies are not always consistently applied to all incoming traffic received by web applications. Failure to check for valid data types of incoming traffic can result in buffer overflows, or other application security breaches. Further, these approaches may not reflect the dynamic behavior of the application traffic. In addition, the security policies are often overly broad. For example, if a policy specifies that "String" is a data type of a password field, then any alphanumeric value entered into the password field will satisfy this data type. However, if a policy specifies that "INT" is a data type of the password field, then only integers can be entered into the password field. Thus, data type "String" is a more broad (or less restrictive) data type than "INT". Requiring a data type of the password field to be "INT" instead of "String" reduces a number of options presented to an intruder with respect to guessing a proper data type of the password field. Requiring a data type of the password field to be "String", in contrast, provides ample opportunities for an intruder to correctly guess the data type of the password field, thereby increasing a number of illegitimate messages entering a web application.

[0008] Accordingly, what is needed is a technique that automatically learns characteristics of the application behavior to generate rules that would prevent malicious traffic from passing through application proxies and gateways. What is also needed is a technique that defines a data type of a message component as narrowly as possible, thereby making it more difficult for an intruder to perpetrate an attack while conforming to the proper data type of a message component.

SUMMARY OF THE INVENTION

[0009] A security gateway receives messages and extracts components thereof, typically in the form of field name-value pairs. The security gateway determines a data type of the values for individual field names to infer the most restrictive data type of the values for that field. The security gateway may then generates rules, which would block messages that do not have values that match the most restrictive data type. Since the most restrictive data type defines a data type of values for the field as narrowly as possible, the generated rules will make it more difficult for an intruder to generate a successful attack while confirming to a valid data type of a value. For example, if the gateway only allows values of an integer data type to pass, an attack that relies on buffer overflows by sending a large amount of shellcode in a field will not succeed because the data type of the shellcode does not match the required integer data type.

[0010] Since messages that have values that do not match the most restrictive data type are likely to represent malicious attacks, the more narrowly the data type of values is defined, the greater the number of illegitimate messages that will be blocked from passing through application proxies and gateways. Accordingly, the generated rules will prevent a maximum number of illegitimate messages from passing through application proxies and gateways.

[0011] In one embodiment, initially, messages are received by a learning engine. Examples of the received messages are URL requests, forms, and other client-server communications. The learning engine extracts message components, such as URL components and field name-value pairs. In the case of field-type messages, such as XML-formatted messages, values for the same component/field name for the messages in the sample are provided as an input to a scalar data type inferencing algorithm as a set of keywords (k). The algorithm determines for a set of keywords a most restrictive data type of the values.

[0012] The algorithm creates a Directed Acyclic Graph (DAG) of a data type set. Examples of the data types in a data type set are known data types, such as "Date," "Time," "INT," "String," and user-defined data types, such as "Word," "URLPATH", and "Credit Card." Some data types in a data type set are in a "contains" type relationship such that data type Di is contained within Dj if every keyword of type Di is also of type Dj. It is said that Di is a more restrictive data type than Dj. The containment relationships provide an ordering amongst at least some of the data types.

[0013] Each node in the DAG is associated with a data type in the data type set. The algorithm traverses the DAG using a breadth-first search algorithm to select a node associated with a most restrictive data type for a given set of keywords.

[0014] The algorithm then creates a list of candidates for a most restrictive data type. Initially, this list is empty. The algorithm then creates a queue using a breadth-first search algorithm and populates the queue with the first data type from the DAG. The algorithm determines whether the first data type is an eligible candidate. To this end, the algorithm determines a match factor of the first data type. If the match factor exceeds a threshold, then the data type is an eligible candidate. Otherwise, the algorithm populates the queue with a next data type.

[0015] The algorithm then looks at all child data types of the eligible data type and computes a match factor for each child data type. The match factor is a fraction of keywords (field names) in a set of keywords that have values that match a particular data type. For example, if there are 100 instances of values for the field name "Password" and 60 of them are strings, and 40 of them are integers, then the match factor for the string data type is 0.60, and the match factor for the integer data type is 0.40.

[0016] If a match factor of the child data type of the eligible data type exceeds a threshold, then the eligible data type is not added to the list of candidate. If the eligible data type has no children with a match factor exceeding a threshold, then the eligible data type is added to the list of candidates. If the child data type with a match factor exceeding a threshold is not on the list of candidates and is not in the queue, the algorithm adds the child data type to the queue.

[0017] Thus, the algorithm infers a most restrictive data type of the set of keywords by determining data types of the keywords. Once the most restrictive data type for values of the same field name is identified, different applications may be made of the data type information. In one embodiment, the learning engine generates a rule, which would allow messages having values for a field name that match the most restrictive data type to pass through the security gateway. The learning engine provides the generated rules to a message filter, which applies the rules to determine whether to allow messages.

[0018] In an alternative embodiment, when the learning engine receives URL messages for analyzing, the learning engine extracts URL components from the messages. The learning engine determines, for URL components at the same level, with the same root URL component, a most restrictive data type of the URL component. The learning engine determines a most restrictive data type using a scalar data type inferencing algorithm. The learning engine generates a rule which would allow messages with the URL components that match the most restrictive data type.

BRIEF DESCRIPTION OF THE DRAWINGS

[0019] Fig. 1 is a block diagram of environment in which the invention operates;

[0020] Fig. 2 is a block diagram of functional components of a security gateway shown in Fig. 1;

[0021] Fig. 3 is a flow chart of a method performed by the security gateway according with one embodiment

[0022] Fig. 4 is a diagram of a Directed Acyclic Graph of a data type set according to an embodiment of the present invention.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

1. System Architecture Overview

[0023] Fig. 1 is a block diagram of an environment 100 in which the invention operates. The environment 100 includes a client system 110 and a server system 120 connected by a communication network 150. A security gateway 130 is interposed between and connected to client 110 and server 120. Although this invention is described in the context of the client-server architecture, it should be understood that the invention can be implemented in any other suitable architecture, such as a peer-to-peer architecture where each system 110 and 120 can be a client and a server.

[0024] As used herein, the term "server system" 120 simply denotes those aspects of the software program associated with a remote computer, as well as underlying operating system and hardware support. As will be understood by those of skill in the art, a server system 120 within the context of the present invention can comprise components of the software program, as well as components of the operating system of a remote computer and hardware components of a remote computer. Server system 120 may include a web infrastructure, such as a web server 140, an application server 160, and a database 170. Web server 140 is a computer running software for serving content across the Internet, such as for example Microsoft Internet Information Server (IIS), or Apache. Web server 140 responds to requests received from client system 110 by transmitting HTML pages across the Internet for display by a web browser (not shown) that is running on client system 110. Application server 160 is a program that handles all application operations between users an enterprise's backend business applications or databases. Database 170, in turn, stores all data relevant to the enterprises' operations. The server system 120 may comprise one or more computers for implementing the above described functionality and services.

[0025] As used herein, the term client system 110 simply denotes those aspects of the software program associated with a user's computer, as well as underlying operating system and hardware support. As will be understood by those of skill in the art, a client system 110 within the context of the present invention can comprise components of the software program, as well as components of the operating system of a user's computer and hardware components of a user's computer. Client system 110 includes a web browsing functionality. While FIG. 1 illustrates a single client system 110, it is appreciated that in actual practice there will be any number of client systems 110 that communicate with the server system 120.

[0026] Communication network 150 can be the Internet, and particularly, the World Wide Web portion thereof. However, it will also be appreciated that communication network 150 can be any known communication network.

[0027] In one implementation, security gateway 130 operates as a proxy in that it is installed directly in front of server 120. In yet another implementation, security gateway 130 operates as as a gateway between the public Internet and an internal network (not shown), such as a WAN, or LAN.

[0028] Security gateway 130 runs the following modules: a message filter 210, a learning engine 230, and memory 250. As used herein, the term "module" refers to computer program logic and/or any hardware to provide the functionality attributed to the module. These modules are described in more detail in reference to Fig. 2.

[0029] Referring now to Fig. 2, message filter 210 is adapted to perform filtering of incoming messages. As described above, examples of the received messages are a web form, a URL, and other client-server communications. Messages can include components, such as cookies, form fields, hidden fields, and URL components. Message filter 210 maintains a list of filtering rules. Message filter 210 processes rules in a list form, checking messages against the rules in order, until the message is either accepted or rejected. If no rule applies, then the default policy (either reject or accept) is applied.

[0030] In one implementation, the learning engine 230 analyzes messages accepted by message filter 210. These messages, for example, can be stored in memory 250 and provided to learning engine 230 to find a most restrictive data type. Learning engine 230 extracts components from the messages. Learning engine 230 determines a most restrictive data type for the extracted components using a scalar data type inferencing algorithm. Learning engine 230 then generates rules, which would allow messages having the components that match the most restrictive data type to pass through the security gateway 130. Learning engine 230 provides the rules to message filter 210. Message filter 210 applies generated rules to determine whether to allow messages with components that match the most restrictive data type to pass through security gateway 130.

[0031] Memory 250 stores a number of data structures. Memory 250 stores various thresholds, match factors, a message log, and a set of data types. The data types stored in memory will be described in more detail below in the "Methods of Operation" section.

2. Methods of Operation

[0032] Referring now to FIG 3, there is shown a flowchart of a method of operation in accordance with one embodiment of the invention.

[0033] It should be noted that security gateway 130 can operate in a learning mode and an operating mode. Briefly, in the learning mode, security gateway 130 receives messages, extracts message components, and determines a most restrictive data type for the extracted message components. Security gateway 130 may then generates rules, which would allow messages having the components that match the most restrictive data type to pass through security gateway 130 and provides the rules to message filter 210. In the operating mode, security gateway 130 applies the rules to determine whether to allow messages with components that match the most restrictive data type.

[0034] Initially, learning engine 230 receives 310 messages and extracts 320 components thereof. Examples of the received messages are web forms, URL requests, and other client-server communications. Examples of message components are URL components and field name-pair values. Although the method of operation will be described in the context of XML-formatted messages, it should be understood that this description applies to other types of messages that include field-type components.

[0035] When learning engine 230 receives XML-formatted messages, it extracts components, such as field name-value pairs. For example, in the XML format, a field name-value pair is: <City_Name> San Francisco </City_Name>. Learning engine 230 extracts the field name "City_Name" and value "San Francisco." Table 1 below shows exemplary field name-value pairs extracted from a set of XML messages. In Table 1 extracted pairs are sorted by the field name.

Table 1. Exemplary Field Name-Value Pairs

Field Name	Value
City_Name	San Francisco
City_Name	Moscow
City_Name	1234
City_Name	New York
City_Name	Paris
First_Name	Jane
First_Name	03/02/70
First_Name	Alex
First_Name	353
First_Name	Ken

[0036] Values for the same component/field name are provided 330 as an input to a scalar data type inferencing algorithm a set of keywords (k). Learning engine 230 executes the data scalar type inferencing algorithm to determine, for a set of keywords, a most restrictive data type for the field name.

A. Identifying a Most Restrictive Data Type For a Set of Keywords

[0037] In the example shown in Table 1, values for the component/field name "City Name" will be provided to the algorithm as a set of keywords: {"San Francisco," "Moscow," "1234", "New York", "Paris"}. Each keyword is a scalar object, e.g., a finite string of characters. Similarly, values for the component/field name "First Name" will be provided to the algorithn as a set of keywords: {"Jane," "03/02/70," "Alex," "353," "Ken"}.

[0038] At this point it is useful to define data types. A data type can be defined as a classification of data representation. Datatypes = {D1, D2, D3 ... Dm} is a set of datatypes. Some data types in the set of data types are known data types, such as "INT" for Integer, "CHAR" for character, "String", "Date", and "Time." Other data types are user-defined data types, such as "URLPATH", "Word", and "Credit Card".

URLPATH

Each URL string, such as http://www.cnn.com/2004/Jan/1/sports.html is a combination of the protocol, such as "http://", a host domain name, such as "www.cnn.com", and a directory path, such as "2004/Jan/1/sports." URLPATH is a data type that describes a directory path of a typical URL.

Word

"Word" data type is defined as a string of alphabetical characters [a-z], which is not interrupted by space or any character other than [a-z].

Credit Card

"Credit Card" is defined as a string of 16 integers.

[0039] Each datatype D is a predicate that takes a keyword as an input, and either accepts or rejects it. D(k) = True, if keyword k is of datatype D, false otherwise.

For example:

INT ("29")=True

INT ("dog")=False

Word ("dog")=True.

[0040] It is said that D(k) is true if the keyword (k) is mapped to the data type D. An alternative way of saying that D(k)=True is that K matches data type D.

[0041] Data types in a set of data types can be in a partially ordered relationship, such as a "contains" type relationship. The "contains" type relationship between two data types creates a hierarchical relationship between the two data types, such a parent-child relationship. It is said that data type Di is contained within Dj if every keyword of type Di is also of type Dj. It is said that Di is a more restrictive data type than Dj. For example, data types "Date" and "String" are in the "contains" type relationship, because every keyword that is of data type "Date" is also of data type "String." Data type "String" is a parent of data type "Date." Data type "Date," in turn, is a child of data type "String." "Date" is a more restrictive data type than "String" and data type "String" is a less restrictive data type than "Date."

[0042] It should be noted that not all data types can be in a "contains" type of relationship. For example, not all keywords of data type "Date" are of data type "INT". Similarly, not all keywords of data type "INT" are of data type "Date." Therefore, these data types are not in a hierarchichal relationship.

[0043] The algorithm uses two new data types "Universe" (U) and "Null" and adds them to the set of data types. Data types "U" and "Null" have the following constraints:

- U(k)=True for all keywords in a data set. In other words, all keywords in a set are of data type U; and
- Null (k)=False for all data types. No keywords in a set are of data type "Null." Adding new data types U and Null expands the set of data types to be {U, D1, D2, D3 ... Dm, Null}.

[0044] The algorithm creates a Directed Acyclic Graph (DAG) of the data types in a data type set, using a "contains" relationship, as shown in Fig. 4. A graph is a finite set of vertices (or nodes) connected by links called edges (or arcs). A DAG is a graph with no path which starts and ends at the same node and repeats no other node. Nodes in a DAG can be in an ancestor – descendant relationship. A DAG has a root node, which is the topmost node that has no ancestor nodes. A DAG has leaf nodes and internal nodes. An internal node is a node that has one or more child nodes. A leaf node has no children. All children of the same node are siblings.

[0045] Referring now to Fig. 4, each node in a DAG is associated with a data type in the set of data types. In Fig. 4, nodes in the DAG are associated with the following data types: "U," "Credit Card," "Date," "INT", "String", "Word", and "Null." It should be noted that the DAG shown in Fig. 3 can include any number of data types. One skilled in the art would understand that the data types shown in Fig. 4 are only exemplary data types.

[0046] As shown in Fig. 4, a node associated with the data type "U" is a root node because it does not have a parent node. A node associated with data type "Null" is a leaf node because it does not have child nodes. The rest of the nodes are internal nodes.

[0047] In any DAG, there is a directed path from data type Dj to data type Di if Dj contains Di. As shown in Fig. 4, data types "String" and "Credit Card" are in a "contains" relationship because every keyword that is of data type "Credit Card" is also of data type "String." It is also said that "Credit Card" is a more restrictive data type than "String."

[0048] The algorithm traverses the DAG using a breadth-first search algorithm to select a node associated with a most restrictive data type for a given set of keywords. The breadth-first search algorithm is a search algorithm that explores all child nodes before exploring descendants of the first child node.

[0049] Initially, the algorithm creates a queue of all data types in the DAG based on the breadth-first search algorithm and populates the queue with the first data type. The algorithm then creates a list of candidates. Each candidate on the list is a most restrictive data type. It should be noted that initially the list of candidates is empty. The list is populated as the most restrictive data types are identified.

[0050] The algorithm then dequeues the first data type from the queue and determines whether the first data type is eligible potential candidate. The data type is eligible if its match factor exceeds a threshold. A match factor is a fraction of keywords in the set of keywords that are of a particular data type. The match factor is computed by iterating over the keywords and counting a number of keywords that match a particular data type. The match factor is a fraction of this number over the total number of keywords in the set. In one implementation, to determine whether a keyword matches a particular data type, learning engine 230 makes an operating system call. An operating system determines whether a keyword in a set of keywords is of a particular data type and returns the result to learning engine 230. In another implementation, learning engine 230 runs a type-checking engine (not shown) which determines whether a keyword is of a particular data type.

[0051] If the match factor of the eligible data type does not exceed a threshold, the algorithm populates the queue with a next data type.

[0052] The algorithm looks at all child data types of the eligible data type and computes a match factor for each child data type. If a match factor of the child data type of the eligible data type exceeds a threshold, then the eligible data type is not added to the list of candidate. If the child data type with a match factor exceeding a threshold has not been previously considered, e.g., the child data type is not on the list of candidates and is not in the queue, the algorithm then adds the child data type to the queue.

[0053] If the eligible data type has no children with a match factor exceeding a threshold, then the eligible data type is added to the list of candidates.

[0054] It should be noted that if the list of candidates includes more than one most restrictive data type, the algorithm picks any data type on the list. Alternatively, the algorithm selects the lowest order data type in the DAG; alternatively, the algorithm can select both data types as acceptable alternatives, depending on the particular application for which the data type information is to be used and utilize other input, such as a user input, or the context of variables.

[0055] The above constraints are representative of one known implementation of an algorithm for selecting a most restrictive data type. It should be noted that there are different variations of constraints used by the algorithm.

[0056] At this point it is useful to define a threshold T. T can be any fraction between 0% and 100%. If T=100%, then the most restrictive data type is the data type that matches all keywords in a set.

[0057] The following example illustrates the operation of the algorithm. Assuming the following values for the component/field name "City Name" were provided to the algorithm as a set of keywords: {"San Francisco," "Moscow," "1234," "New York," "Paris"}. This set of keywords includes 5 keywords.

[0058] The algorithm traverses the DAG using the breadth-first search algorithm and populates a queue with data types from the DAG traversed using the breadth-first search algorithm. Thus, the algorithm populates the queue with the root data type "U."

[0059] The algorithm dequeues the first data type to determine whether "U" is an eligible potential candidate to be a most restrictive data type. The Mf(U) =100%. Therefore, "U" is an eligible candidate. The algorithm then calculates a match factor for all children of "U". "U" has one child, "String." Mf (String)=100% because all keywords in the set are of a data type "String." The algorithm compares the Mf to the threshold of, for example, 75%. This means that the match factor of a data type has to be at least 75% in order to be selected. Since Mf (String) exceeds the threshold, "U" is not a potential candidate. The algorithm then adds "String" to the queue.

[0060] The algorithm then calculates a match factor for all children of "String" as follows:

Mf (Word) =80% because four keywords "San Francisco", "Moscow", and "New York" are of a data type "Word." The algorithm places "Word" in the queue.

Mf (INT)=20% because only one keyword "1234" is of a data type "INT".

Mf (Credit Card)=0% because none of the keywords are of data type "Credit Card."

Mf (Date)=0% because none of the keywords are of data type "Date."

[0061] Since Mf(Word) exceeds the threshold, "String" is not added to the list of candidates. "Word" is added to the queue.

[0062] The algorithm then analyzes data type "Word." Since Mf(Word) exceeds the threshold, "Word" is eligible data type. "Word" does not have any children with a match factor exceeding the threshold, therefore this data type is placed 340 in the list of candidates selected as a most restrictive data type for the field from which these values were extracted.

[0063] Data types "INT", "Credit Card", and "Date" are not eligible data types since their Mf individually do not exceed the threshold.

[0064] Thus, the most restrictive data type for a keyword set {"San Francisco," "Moscow," "1234", "New York", "Paris"} is "Word" since at least 75% of keywords match to this data type and it does not have children with a match factor exceeding the threshold. Thus, the most restrictive data type defines a data type of keywords in the set as narrowly as possible.

[0065] Thus, the algorithm takes a set of keywords, determines a data type of keywords in the set, and infers a most restrictive data type by analyzing the data types of the keywords.

[0066] An alternative implementation of the algorithm avoids computation of Mf for each node. In one such implementation, the DAG is traversed from the bottom up so that the root node is data type "Null" and the botton node is "U". The algorithm traverses the DAG using a breadth-first search algorithm to select a node associated with a most restrictive data type for a given set of keywords. The breadth-first search algorithm is a search algorithm that explores all child nodes before exploring descendants of the first child node. Given the "contains" type relationship between some nodes in the DAG, if Di is an ancestor of Dj, then if Dj(k)=True then Di(k)=True. Thus, according to this embodiment, if a keyword matches a particular data type, determining whether the keyword matches an ancestor of that data type is a redundant operation.

[0067] Continuing with the same example, in this embodiment, the algorithm determines Mf(Word)=80%, which exceeds the threshold of 75%. Therefore, the algorithm does not traverse to the parent node of "Word", "String," to determine the Mf for "String." This embodiment determines the most restrictive data type with the minimum number of computations.

B. Rules Generation

[0068] Given the inferencing of a data type, a number of different applications of this information may be provided. In one embodiment, the data type information is used to filter incoming messages by determining whether the values for the fields have the appropriate data type. Again it should be appreciated that this data type checking does not rely on any apriori knowledge about the data types of each field of a message, but instead on the inferenced data type information. Thus, in this embodiment, once the most restrictive data type for the component is identified, learning engine 350 generates rules, which would allow messages having the components that match the most restrictive data type to pass through security gateway 130. Since the most restrictive data type defines a data type of components as narrowly as possible, the generated rules make it more difficult for an intruder to guess a proper data type of a component. Since the most restrictive data type defines a data type of values for the each field as narrowly as possible, the generated rules will make it more difficult for an intruder to guess a valid data type of a value. Since messages that have values that do not match the most restrictive data type are likely to represent malicious attacks, the more narrowly the data type of values is defined, the greater the number of illegitimate messages that will be blocked from passing through application proxies and gateways. Accordingly, the generated rules will prevent a maximum number of illegitimate messages from passing through application proxies and gateways.

[0069] Continuing with the same example, the algorithm identified data type "Word" as the most restrictive data type of component "CityName." The generated rule is shown below in Table 2:

Table 2. Rule for Type-Field Messages

Field Name	Most Restrictive Data Type
City Name	Word

A rule may be constructed such as IF !(WORD(City_Name)) THEN REJECT

[0070] Learning engine 230 provides the generated rules to message filter 210. Message filter 210 applies the rules to determine whether to allow messages having values of the same field name that match the most restrictive data type.

[0071] It should be understood that although the algorithm was described in the context of field-type messages, such as XML formatted messages, this algorithm is applicable to data type inferencing of components of various types of messages, such as as URL requests. Below is the description of the method of inferencing data types for URL messages.

URL Messages

[0072] At this point it will be useful to explain a structure of a typical URL. Assume that learning engine 230 receives the following URLs:

http://www.cnn.com/2004/Jan/1/sports.html

http://www.cnn.com/2003/Jan/ and

http://www.cnn.com/2002/Dec.

[0073] Each URL string is a combination of the protocol, such as "HTTP://", a host domain name, such as "www.cnn.com", and a directory path, such as 2004/Jan/1/sports. Each of the protocol, host domain name, and directory path are URL components. The directory path has one or more URL components, such as "2004," "Jan," "1," and "sports". The directory path may end with a suffix, such as ".html," which indicates the coded format language used to create hypertext documents. The URL components of a directory path are in a hierarchical relationship with each other. For example, in the URL identified by the path www.cnn.com/2004/Jan/1/sports.html, component "2004" is of a higher level than component "Jan." Similarly, component

"Jan" is of a higher level than component "1." Higher level components are in an ancestor-descendants relationship with lower level components. For example, component "2004" is a parent of component "Jan" and component "Jan" is a child of component "2004." The host domain name component is a root URL component. Components "2004", "2003", and "2002" are components of the same level. Similarly, components "Jan", "Jan", and "Dec" are of the same level.

[0074] Learning engine 230 extracts URL components from the messages. Learning engine 230 determines for URL components at the same level for the selected messages with the same root URL component a most restrictive data type for the URL component. This is performed by using a scalar data type inferencing algorithm described earlier. Learning engine 230 generates a rule which would allow messages with the URL components that match the most restrictive data type.

[0075] Continuing with the same example, the URLs are:

http://www.cnn.com/2004/Jan/1/sports.html

http://www.cnn.com/2003/Jan/ and

http://www.cnn.com/2002/Dec.

[0076] Learning engine 230 extracts the following components:

Level 1: {"2004", "2003", "2002"}

Level 2: {"Jan", "Jan", "Dec"}

Level 3 {"1"}

Level 4 {"sports"}

[0077] Learning engine 230 determines a most restrictive data type for URL components at the same level. For example, to determine a most restrictive data type for the Level 1 components, learning engine 230 provides Level 1 components to the algorithm.

[0078] Referring now to Fig. 4, initially, the algorithm computes a match factor for each data type. The match factor is a fraction of components in the set of components {"2004", "2003", "2002"} that match a particular data type. The result of the computation is as follows:

Mf (U)=100% because all components are of data type "U".

Mf (String)=100% because all components are of data type "String".

Mf (Word) =0% because no component is of data type "Word".

Mf (INT)=100% because all components are of data type "INT".

Mf (Credit Card)=0% because none of the components are of this data type.

Mf (Date)=0% because none of the components are of this data type.

Mf (Null)=0% because no components are of data type "Null".

[0079] The threshold is determined as 75%. Since Mf (INT) exceeds the threshold and "INT" does not have child data types that exceed the threshold, "INT" is the most restrictive data type for the set of components: {"2004", "2003", "2002"}. The generated rule is shown in Table 3 below:

Table 3. Rule for URL messages

URL root component	Level/Most Restrictive Data type
www.cnn.com	Level 1/INT

[0080] For URLs with a root component www.cnn.com, URL components at the first level should match data type "INT". This rule is provided to message filter 210, which will apply the rules to determine whether to allow messages having components that match the most restrictive data type.

[0081] It should be noted that inferencing a data type of a scalar object to find a most restrictive data types has many applications. As previously described, in one application, the algorithm of identifying the most restrictive data type for message components is used to generate rules/constraints that are applied to filter web application traffic.

[0082] In another application, the algorithm is used to generate exception rules to the rules which would otherwise block legitimate traffic, as described in a co-pending application "Using Statistical Analysis to Generate Exception Rules to Allow Legitimate Messages to Pass Through Application Proxies and Gateways," which disclosure is incorporated herein by reference.

[0083] While the algorithm of data type inferencing of scalar objects was described in the context of message filtering, it should be born in mind that the algorithm can be suited in any situation where data needs to be categorized. More generally, the algorithm is beneficially, but not exclusively, used where there is a defined set of types, topics, categories to be applied to a set of data, such as set of labels, keywords, or other entities, where the topics etc. have at least a partial ordering relationship, where the mapping of types to the labels is not known a priori, such as from an external category definition or mapping, and where the labels are associated with data values that may vary over time. The algorithm is applicable where the data is being received serially over time, or where the data is being analyzed in batch. Those of skill in the art will appreciate the wide variety of different applications which have these characteristics.

[0084] The present invention has been described in particular detail with respect to the two possible embodiments. Those of skill in the art will appreciate that the invention may be practiced in other embodiments. First, the particular naming of the components, capitalization of terms, the attributes, data structures, or any other programming or structural aspect is not mandatory or significant, and the mechanisms that implement the invention or its features may have different names, formats, or protocols. Further, the system may be implemented via a combination of hardware and software, as described, or entirely in hardware elements. Also, the particular division of functionality between the various system components described herein is merely exemplary, and not mandatory; functions performed by a single system component may instead be performed by multiple components, and functions performed by multiple components may instead performed by a single component.

[0085] Some portions of above description present the features of the present invention in terms of algorithms and symbolic representations of operations on information. These algorithmic descriptions and representations are the means used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art. These operations, while described functionally or logically, are understood to be implemented by computer programs. Furthermore, it has also proven convenient at times, to refer to these arrangements of operations as modules or by functional names, without loss of generality.

[0086] Unless specifically stated otherwise as apparent from the above discussion, it is appreciated that throughout the description, discussions utilizing terms such as "processing" or "computing" or "calculating" or "determining" or "displaying" or the like, refer to the action and processes of a computer system, or similar electronic computing device, that manipulates and transforms data represented as physical (electronic) quantities within the computer system memories or registers or other such information storage, transmission or display devices.

[0087] Certain aspects of the present invention include process steps and instructions described herein in the form of an algorithm. It should be noted that the process steps and instructions of the present invention could be embodied in software, firmware or hardware, and when embodied in software, could be downloaded to reside on and be operated from different platforms used by real time network operating systems.

[0088] The present invention also relates to an apparatus for performing the operations herein. This apparatus may be specially constructed for the required purposes, or it may comprise a general-purpose computer selectively activated or reconfigured by a computer program stored on a computer readable medium that can be accessed by the computer. Such a computer program may be stored in a computer readable storage medium, such as, but is not limited to, any type of disk including floppy disks, optical disks, CD-ROMs, magnetic-optical disks, read-only memories (ROMs), random access memories (RAMs), EPROMs, EEPROMs, magnetic or optical cards, application specific integrated circuits (ASICs), or any type of media suitable for storing electronic instructions, and each coupled to a computer system bus. Furthermore, the computers referred to in the specification may include a single processor or may be architectures employing multiple processor designs for increased computing capability.

[0089] The algorithms and operations presented herein are not inherently related to any particular computer or other apparatus. Various general-purpose systems may also be used with programs in accordance with the teachings herein, or it may prove convenient to construct more specialized apparatus to perform the required method steps. The required structure for a variety of these systems will be apparent to those of skill in the, along with equivalent variations. In addition, the present invention is not described with reference to any particular programming language. It is appreciated that a variety of programming languages may be used to implement the teachings of the present invention as described herein, and any references to specific languages are provided for disclosure of enablement and best mode of the present invention.

[0090] The present invention is well suited to a wide variety of computer network systems over numerous topologies. Within this field, the configuration and management of large networks comprise storage devices and computers that are communicatively coupled to dissimilar computers and storage devices over a network, such as the Internet.

[0091] Finally, it should be noted that the language used in the specification has been principally selected for readability and instructional purposes, and may not have been selected to delineate or circumscribe the inventive subject matter. Accordingly, the disclosure of the present invention is intended to be illustrative, but not limiting, of the scope of the invention, which is set forth in the following claims.